

Distributed Coverage Maximization via Sketching

MohammadHossein Bateni
Google Research

Hossein Esfandiari
University of Maryland

Vahab Mirrokni
Google Research

Abstract

Coverage problems are central in optimization and have a wide range of applications in data mining and machine learning. While several distributed algorithms have been developed for coverage problems, the existing methods suffer from several limitations, e.g., they all achieve either suboptimal approximation guarantees or suboptimal space and memory complexities. In addition, previous algorithms developed for submodular maximization assume oracle access, and ignore the computational complexity of communicating large subsets or computing the size of the union of the subsets in this subfamily. In this paper, we develop an improved distributed algorithm for the k -cover and the set cover with λ outliers problems, with almost optimal approximation guarantees, almost optimal memory complexity, and linear communication complexity running in only four rounds of computation. Finally, we perform an extensive empirical study of our algorithms on a number of publicly available real data sets, and show that using sketches of size 30 to 600 times smaller than the input, one can solve the coverage maximization problem with quality very close to that of the state-of-the-art single-machine algorithm.

1 Introduction

As important special cases of submodular optimization, maximum coverage and minimum set cover are among the most central problems in optimization with a wide range of applications in machine learning, document summarization, and information retrieval (e.g., see [12, 2, 11, 27, 6]). These problems, referred to as *coverage* problems, are defined as follows: consider a ground set \mathcal{E} of m elements, and a family $\mathcal{S} \subseteq 2^{\mathcal{E}}$ of n subsets of the elements (i.e., $n = |\mathcal{S}|$ and $m = |\mathcal{E}|$).¹ The *coverage function* \mathcal{C} is defined as $\mathcal{C}(S) = |\cup_{U \in S} U|$ for any subfamily $S \subseteq \mathcal{S}$ of subsets. In the k -cover problem, given a parameter k , the goal is to find k sets in \mathcal{S} with the largest union size. In the *set cover* problem, the goal is to pick the minimum number of sets from \mathcal{S} such that all elements in \mathcal{E} are covered. In this paper, we also consider the following variant of the set cover problem, called the *set cover with λ outliers* problem², where the goal is to find the minimum number of sets covering at least a $1 - \lambda$ fraction of the elements in \mathcal{E} .

Solving coverage problems on huge data sets becomes increasingly important for various large-scale data mining and machine learning applications, and there is a clear need to develop distributed solutions to these problems [12, 13, 10, 20, 27, 17, 24]. In the distributed computation model, we assume that the data is distributed across multiple machines. A distributed algorithm runs in rounds. In each round, the data is processed in parallel on all machines. In particular, each machine waits to receive all messages sent to it in the previous round. Then it performs its own

¹There are two separate series of work in this area. We use the convention of the submodular/welfare maximization formulation [5], whereas the hypergraph-based formulation [28] typically uses n, m in the opposite way.

²This is sometimes called the $(1 - \lambda)$ -partial cover problem in the literature.

Table 1: Comparison of our results to prior work.

Problem	Credit	# rounds	Approximation	Load per machine	Comment
k -cover	[20]	$O(\frac{1}{\varepsilon\delta} \log m)$	$1 - \frac{1}{e} - \varepsilon$	$O(mkn^\delta)$	submod. functions
k -cover	[24]	2	0.54	$\max(mk^2, mn/k)$	submod. functions
k -cover	[14]	$\frac{1}{\varepsilon}$	$1 - \frac{1}{e} - \varepsilon$	$\frac{\max(mk^2, mn/k)}{\varepsilon}$	submod. functions
k -cover	Here	4	$1 - \frac{1}{e} - \varepsilon$	$\tilde{O}(n)$	-
Set cover with outliers	Here	4	$(1 + \varepsilon) \log \frac{1}{\lambda}$	$\tilde{O}(n)$	-

computation, after which it has a chance to send messages to the other machines. The MapReduce framework [16] easily fits into this model. The load on each machine is the total data it processes. In this model, the load of a machine should be sublinear in terms of the input size. In fact, two important factors determining the performance of a distributed algorithm in this model are (i) the number of rounds of computation, and (ii) the maximum load on any machine. These parameters have been discussed and optimized in previous work discussing distributed algorithms in MapReduce framework [19, 17, 3, 7, 24].

While many techniques have been developed to solve the more general class of submodular maximization problems in a distributed manner [10, 20, 27, 6, 24, 15], they do not take advantage of the special structure of coverage functions, and as a result, they achieve either suboptimal approximation guarantees or suboptimal space and memory complexities in terms of the number of (input) subsets involved. Moreover, most previous results for submodular maximization either explicitly or implicitly assume that one has *value oracle access* to the submodular function. Such an oracle for coverage functions has the following form: given a subfamily of (input) subsets, determine the size of the union of the subsets in the subfamily. Implementing this for coverage functions is not practical in the presence of a large ground set \mathcal{E} and/or large subsets in the instance. In particular, assuming that we can communicate the whole subsets across machines in a distributed setting overlooks the complexity of sending large subsets around.

Our Contributions. In this paper, we aim to develop distributed algorithms that achieve almost optimal approximation guarantees for coverage problems in distributed models with optimal space and memory complexities without assuming value oracle access. To do so, we employ a sketching technique that we developed recently [8]. Our contributions in this paper are three-fold. First of all, we develop a distributed algorithm for k -cover and set cover with λ outliers, which are almost optimal from three perspectives: (i) they achieve optimal approximation guarantees of $1 - 1/e$ and $\log \frac{1}{\lambda}$ for the above two problems, respectively; (ii) they have a memory complexity of $O(n)$ and also linear communication complexity; and finally (iii) they run in a small constant (i.e., four) rounds of computation. Table 1 provides a brief comparison of our theoretical results and the previous work. Secondly, since we do not assume value oracle access to the coverage function, our algorithms for coverage problems give us flexibility to apply our techniques for the graph-theoretic dominating set problem (which is closely related to coverage problems). Our algorithm is the first such distributed algorithm for the dominating set problem without the requirement to load all edges connected to a node on a machine. This is crucial for handling graphs with very high-degree nodes. Finally, we

show the power of our techniques by performing an extensive empirical study of our algorithms on a number of publicly available real data sets. We observe that sketches that are 30 to 600 times smaller than the input size suffice for solving the coverage maximization problem with quality very close to that of the state-of-the-art single-machine algorithm; e.g., for a medium-size data set that we can run the single-machine stochastic-greedy algorithm, we can achieve 99.6% of the quality of the single-machine stochastic-greedy algorithm for maximum coverage using only 3% of the data.

Related Work. Being a notable special case of submodular maximization, solving maximum k -coverage in a distributed manner has attracted significant amount of research over the last few years [12, 13, 21, 10, 20, 27, 6, 24, 15, 26]. In all these papers, oracle access to the submodular function is assumed, and therefore, the running time of each round of the algorithms depends on the size of the sets as well. In this model, for the coverage maximization problem, Chierichetti et al. [12] present a $(1 - 1/e)$ -approximation algorithm in polylogarithmic number of rounds of computation, and this was improved to $O(\log n)$ rounds [10, 20]. Recently, applying the idea of randomized core-sets, a constant-approximation algorithm has been developed for this problem that runs in 2 rounds [24, 15]. In particular, the best known approximation factor for this model is 0.54 [24]. Recently, a distributed algorithm for the submodular cover (a generalization of set cover) has been presented in the MapReduce framework [26], but the distributed model is slightly different and the number of rounds is superconstant (and superlogarithmic).

More Notation. Coverage problems can also be modeled as a bipartite graph G . In this graph, \mathcal{S} corresponds to one part of vertices of G , and \mathcal{E} corresponds to the other part. For each set $S \in \mathcal{S}$, there are $|S|$ edges in G from the vertex corresponding to S to vertices corresponding to elements $i \in S$. For simplicity, we assume that there is no isolated vertex in \mathcal{E} . For a (bipartite) graph G and a subset S of its vertices, we denote by $\Gamma(G, S)$ the set of neighbors of S in G . When applied to a bipartite graph G modeling a set-cover instance, we can write the coverage problem as $\mathcal{C}(S) = |\Gamma(G, S)|$ for any $S \subseteq \mathcal{S}$.

2 Distributed Algorithm for Coverage Problems

In this section we present distributed algorithms for k -cover and set cover with λ outliers. Our algorithms need only $\tilde{O}(n)$ space per machine.

We say an algorithm is sketch-based if it first constructs a sketch from the input graph, and then it accesses the sketch to solve the problem without looking at the input graph directly again. In a recent work [8] we provide offline sketch-based streaming algorithms for the k -cover problem and the set cover with λ outliers problem. Those results rely on a powerful sketch, denoted by $H_{\leq n}$. The Algorithm 1 shows a construction for $H_{\leq n}$ ³. The algorithm for k -cover constructs the sketch $H_{\leq n}$, and then simply solves the problem on it. The algorithm for set cover with λ outliers makes a logarithmic number of guesses about the size of the solution, constructs $H_{\leq n}$ sketches for each, and then solves the problem on each resulting sketch. The focus in the previous work [8] is on a streaming implementation, however, the same would work if the sketch can be constructed in any other computational model.

³We drop parameters k, ε , and δ'' whenever they are clear from the context.

Theorem 1 (From [8]). *For any $\varepsilon \in (0, 1]$ and any graph G (modeling a set-cover instance), there exist sketch-based algorithms that succeed with probability $1 - \frac{1}{n}$ in finding the following.*

1. *One finds a $(1 - \frac{1}{e} - \varepsilon)$ -approximate solution to k -cover on G , working on a sketch that consists of $\tilde{O}(n)$ edges.*
2. *The other, given $C \geq 1$, finds a $(1 + \varepsilon) \log \frac{1}{\lambda}$ approximate solution to set cover with λ outliers on G . The sketches used in this algorithm have $\tilde{O}(n/\lambda^3) = \tilde{O}_\lambda(n)$ edges in total.*

Here, we provide a nontrivial construction of sketch $H_{\leq n}$ in four rounds of computations.⁴ Then, we apply Theorem 1 to solve k -cover and set cover with λ outliers in the fourth round.

Algorithm 1 $H_{\leq n}(k, \varepsilon, \delta'')$

Input: An input graph G as well as parameters $k, \varepsilon \in (0, 1]$, and δ'' .

Output: Sketch $H_{\leq n}(k, \varepsilon, \delta'')$.

- 1: Set $\delta = \delta'' \log \log_{1-\varepsilon} m$.
 - 2: Let h be an arbitrary hash function that uniformly and independently maps \mathcal{E} in G to $[0, 1]$.
 - 3: Initialize $H_{\leq n}(k, \varepsilon, \delta'')$ with vertices \mathcal{S} of G , and no edge.
 - 4: **while** number of edges in $H_{\leq n}(k, \varepsilon, \delta'')$ is less than $\frac{24n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}$ **do**
 - 5: Pick $v \in \mathcal{E}$ of minimum $h(v)$ that is still not in $H_{\leq n}(k, \varepsilon, \delta'')$.
 - 6: **if** degree of v in G is less than $\frac{n \log(1/\varepsilon)}{\varepsilon k}$ **then**
 - 7: Add v along with all its edges to $H_{\leq n}(k, \varepsilon, \delta'')$.
 - 8: **else**
 - 9: Add v along with $\frac{n \log(1/\varepsilon)}{\varepsilon k}$ of its edges, chosen arbitrarily, to $H_{\leq n}(k, \varepsilon, \delta'')$.
-

Lemma 2. *Pick arbitrary graph G as well as a number k , and parameters $\varepsilon \in (0, 1]$ and $\delta'' \in (0, 1]$. Let $H = H_{\leq n}$ be the sketch reported by Algorithm 1. Then with probability $1 - 1/n^2$,*

- *the hash values of all elements in H is at most $\frac{2\tilde{n}}{m}$, and*
- *there are at most $3\tilde{n}$ edges with hash value $\frac{2\tilde{n}}{m}$,*

where $\tilde{n} = \frac{24n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}$.

Proof. Note that Algorithm 1 stops adding elements to $H_{\leq n}$ as soon as the number of edges in this sketch hits \tilde{n} . Thus, excluding the last element of $H_{\leq n}$, there are less than \tilde{n} edges in $H_{\leq n}$, hence less than \tilde{n} elements in this sketch. Therefore, $H_{\leq n}$ contains at most \tilde{n} elements.

In the rest of the proof we show that, with probability $1 - 1/n$, the number of elements with hash value less than $\frac{2\tilde{n}}{m}$ is within the range $[\tilde{n}, 3\tilde{n}]$. The lower bound together with the fact that $H_{\leq n}$ contains at most \tilde{n} elements proves the first part of the theorem. The upper bound directly proves the second part of the theorem.

For every element v , let X_v be the binary random variable indicating whether $h(v) < \frac{2\tilde{n}}{m}$, and let $X = \sum_{v \in \mathcal{E}} X_v$ denote the number of elements with hash value less than $\frac{2\tilde{n}}{m}$. The Chernoff bound gives

$$\Pr\left(|X - \mathbb{E}[X]| \geq \frac{1}{2}\mathbb{E}[X]\right) \leq 2 \exp\left(-\frac{\frac{1}{4}\mathbb{E}[X]}{3}\right) = 2 \exp\left(-\frac{\mathbb{E}[X]}{12}\right). \quad (1)$$

⁴The number of rounds have not been optimized.

Remark that h maps each element to $[0, 1]$ uniformly at random. Thus, the probability of $h(v) \leq \frac{2\tilde{n}}{m}$ for any arbitrary element v is exactly $\frac{2\tilde{n}}{m}$. Therefore, we have

$$\mathbf{E}[X] = \sum_{v \in \mathcal{E}} \mathbf{E}[X_v] = \sum_{v \in \mathcal{E}} \frac{2\tilde{n}}{m} = 2\tilde{n}. \quad (2)$$

Putting Equations (1) and (2) together gives us

$$\begin{aligned} \Pr(|X - 2\tilde{n}| \geq \tilde{n}) &\leq 2 \exp\left(-\frac{\tilde{n}}{6}\right) = 2 \exp\left(-\frac{4n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}\right) \\ &\leq 2 \exp(-2 \log n - 1) < \exp(-2 \log n) = \frac{1}{n^2}. \end{aligned}$$

Thus with probability $1 - \frac{1}{n^2}$ we have $\tilde{n} \leq X \leq 3\tilde{n}$, which completes the proof. \square

Algorithm 2 A distributed algorithm to compute $H_{\leq n}(k, \varepsilon, \delta'')$

Input: An input graph G as well as parameters $k, \varepsilon \in (0, 1]$, and δ'' .

Output: Sketch $H_{\leq n}(k, \varepsilon, \delta'')$.

- 1: **Round 1:** Send the edges of each element to a distinct machine. Let $\tilde{n} = \frac{24n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}$. For each element v , if $h(v) \leq \frac{2\tilde{n}}{m}$, the machine corresponding to v sends $h(v)$ along with its degree to machine number 1; it does nothing otherwise.
 - 2: **Round 2:** Machine number 1 iteratively selects elements with the smallest h until the sum of the degrees of the selected vertices reaches \tilde{n} . This machine informs the corresponding machine of selected elements.
 - 3: **Round 3:** For each selected element v , if the degree of v is less than $\frac{n \log(1/\varepsilon)}{\varepsilon k}$, machine v sends all its edges to machine one. Otherwise, it arbitrarily sends $\frac{n \log(1/\varepsilon)}{\varepsilon k}$ of its edges to machine one.
 - 4: **Round 4:** The subgraph sent to machine one is the sketch $H_{\leq n}(k, \varepsilon, \delta'')$.
-

Theorem 3. *There is a four-round distributed algorithm constructing $H_{\leq n}$ in $\tilde{O}(n)$ space with probability $1 - \frac{1}{n^2}$.*

Proof. Notice that the conditions of Lemma 2 hold with probability $1 - \frac{1}{n^2}$. We prove this theorem given that those conditions hold. We first show that Algorithm 2 consumes $\tilde{O}(n)$ space per machine. Next, we show that the sketch reported by this algorithm is $H_{\leq n}$.

The degree of each element is at most n , the number of sets; thus, the space consumption of each machine in the first and third rounds is $\tilde{O}(n)$. In the second round machine number 1 receives $\tilde{O}(1)$ bits from each machine independently with probability $\frac{2\tilde{n}}{m}$. Using the second condition of Lemma 2, the number of messages that this machine receives is at most $3\tilde{n}$. Therefore, this machine uses $\tilde{O}(n)$ space. The number of edges machine one receives in the fourth round is at most $\tilde{n} + n \in \tilde{O}(n)$.

According to the first condition of Lemma 2, no element in $H_{\leq n}$ has hash value more than $\frac{2\tilde{n}}{m}$. Thus, the corresponding elements of machines that send nothing to machine one in the first round are not in $H_{\leq n}$. Thus, the set of elements that machine one selects is the same as the elements of $H_{\leq n}$. As a result, what machine one receives in the fourth round is $H_{\leq n}$, as desired. \square

Theorem 4. *There exists a four-round distributed algorithm that reports a $(1 - \frac{1}{e} - \epsilon)$ -approximate solution to k -cover, with probability $1 - \frac{2}{n}$. Moreover, no machine uses more than $\tilde{O}(n)$ space in this algorithm.*

Proof. We use Algorithm 2 to construct $H_{\leq n}$. This succeeds with probability $1 - \frac{1}{n^2}$ by Theorem 3. Then, we apply Theorem 1 to solve the problem in the last round. Now this theorem guarantees that our algorithm gives a $(1 - \frac{1}{e} - \epsilon)$ -approximate solution to k -cover, with probability $1 - \frac{1}{n} - \frac{1}{n^2} < 1 - \frac{2}{n}$. \square

Theorem 5. *There exists a four-round distributed algorithm that reports a $(1 + \epsilon) \log \frac{1}{\lambda}$ -approximate solution to set cover with λ outliers, with probability $1 - \frac{2}{n}$. Moreover, each machine uses $\tilde{O}(n)$ space.*

Proof. We run $\log_{1+\epsilon/3} n$ copies of Algorithm 2 to construct $\log_{1+\epsilon/3} n$ different instances of $H_{\leq n}$ required by Theorem 1. Theorem 3 implies that each run of Algorithm 2 constructs its corresponding sketch correctly, with probability $1 - \frac{1}{n^2}$. This together with Theorem 1 proves that our algorithm gives a $(1 + \epsilon) \log \frac{1}{\lambda}$ -approximate solution to set cover with λ outliers, with probability $1 - \frac{1}{n} - \log_{1+\epsilon/3} n \frac{1}{n^2} < 1 - \frac{2}{n}$. \square

3 Dominating set

In the dominating set problem we are given a graph G with the goal of selecting the minimum number of vertices such that every other vertex is the neighbor of one selected. We say a subset S of vertices in G is a dominating set with λ outliers if at most a λ fraction of vertices of G neither exists nor has a neighbor in S . A set of vertices S is an α -approximate solution to dominating set with λ outliers if (1) it is a dominating set with λ outliers of the graph, and (2) $|S|$ is at most α times the size of the minimum dominating set.

The following theorem provides the first distributed algorithm for dominating set with λ outliers.

Theorem 6. *There exists a four-round distributed algorithm that reports a $(1 + \epsilon) \log \frac{1}{\lambda}$ -approximate solution to dominating set with λ outliers, with probability $1 - \frac{2}{n}$. Moreover, each machine use only $\tilde{O}(n)$ space.*

Proof. Here, we give a reduction from dominating set with λ outliers to set cover with λ outliers. Let graph H be an instance of dominating set with λ outliers. We construct an instance of set cover with λ outliers, denoted by G . Each of the n vertices in \mathcal{S} corresponds to a vertex in H . Similarly, the n vertices in \mathcal{E} correspond to vertices in H . We place an edge between $i \in \mathcal{S}$ and $j \in \mathcal{E}$ if and only if $i = j$ or there is an edge between i and j in graph H .

For any vertex v of H , the union of v and its neighbors in H forms the set of elements that the set v covers in G . Similarly the elements that a subset $S \subseteq \mathcal{S}$ covers is exactly the set of vertices in S and all their neighbors. Thus any set cover with λ outliers on G is a dominating set with λ outliers on H , and vice versa. \square

In the k -dominating set problem we are given a graph G and a number k , with the goal of selecting k vertices that maximize the number of vertices that are either selected or one of their neighbors is. A set of vertices S is an α -approximate solution to k -dominating set if it covers α times that of the optimum.

The following theorem provides the first distributed algorithm for k -dominating set.

Theorem 7. *There exists a four-round distributed algorithm that reports a $1 - 1/e - \varepsilon$ -approximate solution to k -dominating set, with probability $1 - \frac{2}{n}$. Moreover, each machine uses only $\tilde{O}(n)$ space.*

Proof. The proof here is similar to the proof of Theorem 6. We give a reduction from k -dominating set to k -cover. Let graph H be an instance of k -dominating set. Similar to the proof of Theorem 6 we construct an instance of k -cover, denoted by G . Once again, each set in \mathcal{S} corresponds to a vertex in H , as is each element in \mathcal{E} . We place an edge between $i \in \mathcal{S}$ and $j \in \mathcal{E}$ if and only if $i = j$ or there is an edge between i and j in graph H .

For any set v in G , the set of elements covered by v is exactly the union of v and all of its neighbors. Similarly, for any subset $S \subseteq \mathcal{S}$, the elements that S covers is the union of vertices in S and all their neighbors. Therefore any k -cover on G is a k -dominating set with the same coverage, and vice versa. \square

4 Empirical Study and Results

We first give an overview of what datasets we use in our empirical study. Afterwards we mention the methodology for our experiments before discussing the individual results. We run our empirical study on four types of instances. A summary is presented in Table 3. *Dominating set* instances include `livejournal-3`, `livejournal-2`, `dblp-3` and `dblp-2`. Here the goal is to cover the nodes via multi-hop neighborhoods. We have two sets of *bag-of-words* instances, where the goal is to cover as many words/bigrams via selecting a few documents: `gutenberg` and `s-gutenberg` for books and `reuters` for news articles. Instances `wiki-main` and `wiki-talk` are our contribution graphs where we want to find a set of users who revised many articles. Finally we have some *planted set-cover* instances that are known to fool the greedy algorithm: `planted-A`, etc.

Dominating set instances. We build set-cover instances based on graphs for LiveJournal (social network) [30] and DBLP (database of coauthorship) [4]. The vertices of these graphs form both \mathcal{S} and \mathcal{E} in the new instances `livejournal-3` and `dblp-3`. A set u covers an element v if and only if v is within the three-hop neighborhood of u in the original graph (i.e., reachable by a path of length no more than three). Similarly, we build two smaller instances `dblp-2` and `livejournal-2` that differ from the previous two, in that they are based on 2-hop neighborhoods. To show scalability further, we also use a sampled version of `livejournal-3`: each vertex is picked with a fixed probability.

“Bag of words” instances. We build `gutenberg` based on documents and bigrams inside them. The starting point is the set of 50,284 books on Project Gutenberg with IDs less than 53,000 [1],⁵ downloaded via [29]. We then remove the English stopwords [9], and throw away 8,568 books we think are not in English, leaving us with 41,716 books. This was done heuristically: any book with more than half its distinct words missing from an English word list [18] was deemed non-English. (Non-English books in the collection make the set-cover instances much simpler, since picking books from different languages allows us to cover a lot of new words/bigrams. This process drops the number of distinct words by about 63%.) Natural Language Toolkit [9] was then used to turn words into their stems, before generating the list of bigrams in each book. A smaller version of this dataset, called `s-gutenberg`, was also constructed using books with IDs less than 1,000. We also consider another bag-of-words instance, `reuters`, which is a collection of Reuters news articles [23]

⁵The upper bound was picked because the project claims to host 52,031 books [1].

Table 2: Parameters used to generated “planted” datasets.

Name	k	m	k'	ϵ
planted-A	100	10,000	10,000	0.2
planted-B	100	1,000,000	100,000	0.2
planted-C	500	10,000,000	100,000	0.2
planted-D	1,000	10,000,000	100,000	0.2

Table 3: General information about our datasets.

Name	Type	$ \mathcal{S} $	$ \mathcal{E} $	$ E $
livejournal-3	dominating set	3,997,962	3,997,962	72,803,204,325
livejournal-2	dominating set	3,997,962	3,997,962	3,377,182,611
dblp-3	dominating set	317,080	317,080	333,505,724
dblp-2	dominating set	317,080	317,080	27,437,914
gutenberg	bag of words	41,716	99,949,091	1,068,977,156
s-gutenberg	bag of words	925	10,620,424	27,337,479
reuters	bag of words	199,328	138,922	15,334,605
planted-A	planted	10,100	10,000	1,220,000
planted-B	planted	100,100	1,000,000	1,201,100,000
planted-C	planted	100,500	10,000,000	2,410,100,000
planted-D	planted	101,000	10,000,000	1,210,100,000
wiki-main	contribution graph	2,953,425	10,619,081	75,151,304
wiki-talk	contribution graph	1,736,343	1,017,617	7,299,920

written 1996–1997. The words in each article have already been changed to their stems. There are four medium-size subdatasets in the collection. We only report results on the `p0` part. The others behave similarly.

Contribution graphs. The Wikipedia edit history (up to 2008) is available on SNAP [22]. We build two datasets `wiki-main` and `wiki-talk` based on this. In either namespace (main article texts or the talk pages), users have made revisions. We place an edge between a user and a page if the user has revised the page. A set-cover solution here consists of a group of users who have revised all (or many) pages.

Planted set-cover instances One can generate instances where a good set cover is planted in an otherwise random graph. The advantage is that even for large instances we know what the optimum solution is. We build such graphs with parameters k , m , k' and ϵ . Such an instance has a ground set of size m and $k + k'$ sets. Out of these, k have the same size and perfectly cover the entire ground set, while the other k' have random elements but are a factor $1 + \epsilon$ larger than the planted sets. We use four such graphs in our experiments with parameters mentioned in Table 2.

4.1 Approach

The construction of the sketch H is based on two types of prunings for edges and vertices of the input graph:

- Subsampling the elements, and

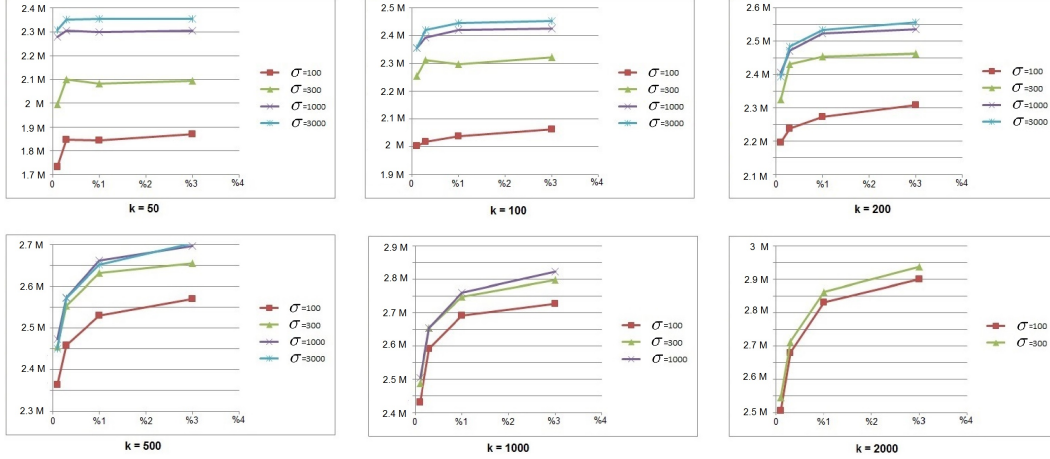


Figure 1: These plot number of covered nodes against the relative size of the sketch on `livejournal-3` with $\rho \in [10^{-3}, 3 \cdot 10^{-2}]$, $\sigma \in [100, 5000]$, and $k \in [10^2, 10^4]$. Curves in one plot correspond to different choices for σ . With large σ , the results of some runs are indistinguishable from the one next to it in the plot, hence invisible.

- Removing edges from large-degree elements.

The theoretical definition of H allows one to compute (i) the probability of sampling an element, and (ii) the upper bound on the degree of the elements. Although these two parameters may be almost tight in theory, in practice one may use smaller values to get the desirable solutions. In this section we parameterize our algorithm by ρ and σ , where ρ is the probability with which we sample an element, and σ is the upper bound on the degree of the elements. We investigate this in our experiments.

The `STOCHASTICGREEDY` algorithm [25] achieves $1 - \frac{1}{e} - \epsilon$ approximation to maximizing monotone submodular functions (hence coverage functions) with $O(n \log(1/\epsilon))$ calls to the submodular function. Theoretically, this is the fastest known $1 - \frac{1}{e} - \epsilon$ approximation algorithm for coverage maximizations, and is shown to be the most efficient in practice for maximizing monotone submodular functions, when the input is very large. We plug this into our MapReduce algorithm and show that our algorithm runs much faster, while losing a very small fraction on the efficiency. For smaller instances we compare our algorithm to `STOCHASTICGREEDY`, but for larger ones we provide convergence numbers to argue that the two should get very similar coverage results.

LiveJournal social network. We try different values for ρ , σ and k when running our algorithm on `livejournal-3`; see Figure 1. When k is small, the result improves as σ grows, but increasing ρ has no significant effect. On the other hand, for larger k , the improvement comes from increasing ρ while σ is not as important. This observation matches the definition of H , in which the bound on the degree is decreasing in k and the sampling rate is increasing in k .

DBLP coauthorship network. Figure 2 shows the results of running the algorithm on `dblp-3` with a range of parameters, while the performance is compared to that of `STOCHASTICGREEDY`. Each point in these plots represents the mean of three independent runs. Interestingly, a sketch with

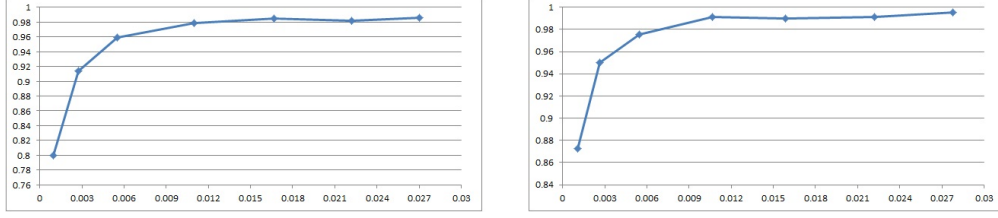


Figure 2: The results for **dblp-3** are shown for $\rho \in [2 \cdot 10^{-3}, 5 \cdot 10^{-2}]$, $\sigma = 100$. With $k = 100$ on the left and $k = 50$ on the right, we plot our performance relative to **STOCHASTICGREEDY** against the fraction of edges from the input graph retained in our sketch.

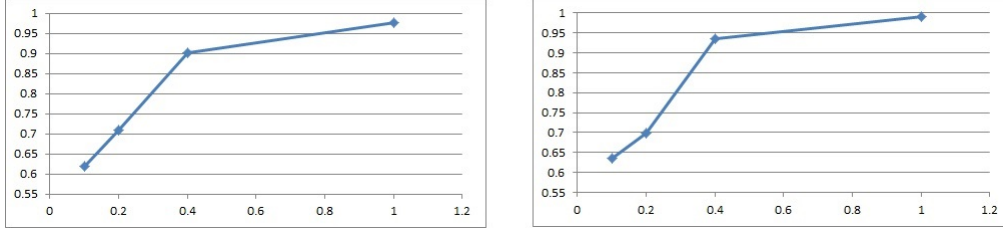


Figure 3: The above are results of running the algorithm on sampled version of **dblp-3** with $\rho = 0.02$, $\sigma = 100$; see $k = 100$ on the left, and $k = 50$ on the right. The x axis size denotes the size of the sampled graph relative to the whole. The y axis shows the quality relative to **STOCHASTICGREEDY**.

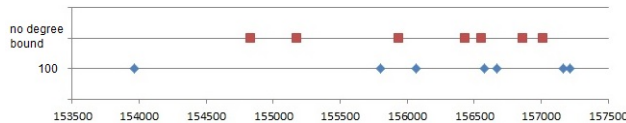
merely 3% of the memory footprint of the input graph attains %99.6 of the quality of **STOCHASTICGREEDY**.

We run our algorithm on induced subgraphs of **dblp-3** with different sizes; see Figure 2. Interestingly, the performance of our algorithm improves the larger the sampled graph becomes. In other words, if one finds parameters ρ and σ on a subgraph of the input and applies it to the whole graph, one does not lose much in the performance.

Figure 4 compares the results of our algorithm for $\sigma = 100$ and no degree bound ($\sigma = \infty$). We observe that picking $\sigma = 100$ does not incur a significant loss in the performance.

Project Gutenberg. We run our algorithm on **gutenberg** with different parameters for ρ and σ . As shown in Figure 5, the outcome of the algorithm converges quickly. In other words, for $\rho = 0.003$ and $\sigma = 100$, the outcome of **STOCHASTICGREEDY** on our sketch and on the input graph are quite similar, while our sketch is 600 times smaller.

Figure 4: Results on **dblp-3** for $\sigma = 100$ (red dots) and $\sigma = \infty$ (blue dots) are depicted. Here we used $\rho = 0.02$ and $k = 50$.



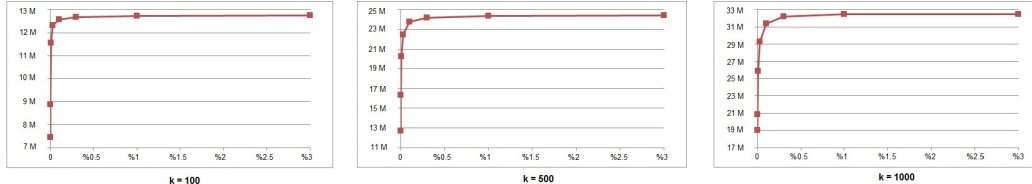


Figure 5: The above plots the number of covered bigrams against ρ for **gutenberg** with $\rho \in [10^{-5}, 3 \cdot 10^{-2}]$, $\sigma \in [10^2, 10^4]$, and $k \in [10^2, 1000^3]$. The curves corresponding to different values of σ are practically indistinguishable.

Other datasets. We now present some results for the datasets we did not get to report on in details. Table 4 shows that for these datasets, a small sketch suffices to get close to the single-machine greedy solution. In fact, these are small enough for the greedy algorithm to run on one machine.

Table 4: Results for other datasets.

Instance	Footprint	Quality	Instance	Footprint	Quality
wiki-main	0.06%	94.4%	reuters	10%	96%
wiki-main	2.4%	99.5%	dblp-2	1.7%	92%
wiki-main	7.7%	99.9%	dblp-2	3.1%	96%
wiki-talk	1.5%	99.2%	reuters	1.2%	87%
planted-A	8.2%	96%	reuters	3.6%	92%

The **livejournal-2** instance is too big for the greedy algorithm to run on a single machine. Here we compare our result to what is achievable for a 10% sample of the instance, which has about 340 million edges. With a 0.8% footprint we can obtain a solution essentially getting the same result. With footprints 0.3%, 0.2%, 0.1% and 0.75%, we lose no more than 1%, 3%, 4% and 9%, respectively.

Except for the smallest, the planted instances are also too big for the greedy algorithm to run. Nonetheless, looking at the numbers, e.g., for **planted-B**, we notice that the quality of the greedy solution is almost the same for sketches of relative sizes 0.3% and 42%—the latter has about 500 million edges. In particular, sketches of relative size 0.3%, 1% and 10% produce 3%, 2% and 1% error, respectively, compared to the sketch of size 42%. The results are similar for the other two planted instances.

5 Conclusions

In this paper, we present an almost optimal distributed algorithm for covering problems. Our algorithm beats the previously developed algorithms in several fronts: e.g., (i) they provably achieve the optimal approximation factors for these problems, (ii) they run in only four rounds of computation (as opposed to logarithmic number of rounds), and (iii) their space complexity is independent of the number of elements in the ground set. Moreover, our algorithms can handle covering problems with huge subsets (in which even one subset of the input may not fit on a single machine). Our empirical study shows the superiority of our algorithms in practice. This technique also applies to

streaming models, and it would be nice to perform a formal empirical study of these algorithms in those models in the future.

References

- [1] Search Project Gutenberg. <https://www.gutenberg.org/ebooks/>. Accessed: 05-19-2016.
- [2] Z. Abbassi, V. S. Mirrokni, and M. Thakur. Diversity maximization under matroid constraints. In *KDD*, pages 32–40, 2013.
- [3] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *STOC*, pages 574–583, 2014.
- [4] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *KDD*, 2006.
- [5] A. Badanidiyuru, S. Dobzinski, H. Fu, R. Kleinberg, N. Nisan, and T. Roughgarden. Sketching valuation functions. In *SODA*, pages 1025–1035. SIAM, 2012.
- [6] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. In *KDD*, 2014.
- [7] M. Bateni, A. Bhashkara, S. Lattanzi, and V. Mirrokni. Mapping core-sets for balanced clustering. In *NIPS*, 2014.
- [8] M. Bateni, H. Esfandiari, and V. Mirrokni. Almost optimal streaming algorithms for coverage problems. *CoRR*, abs/1610.08096, 2016.
- [9] S. Bird. NLTK: The natural language toolkit. COLING-ACL ’06, pages 69–72, 2006.
- [10] G. E. Blelloch, H. V. Simhadri, and K. Tangwongsan. Parallel and I/O efficient set covering algorithms. In *SPAA*, pages 82–90, 2012.
- [11] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS*, pages 155–166, 2012.
- [12] F. Chierichetti, R. Kumar, and A. Tomkins. Max-Cover in Map-Reduce. In *WWW*, pages 231–240, 2010.
- [13] G. Cormode, H. J. Karloff, and A. Wirth. Set cover algorithms for very large datasets. In *CIKM*, pages 479–488, 2010.
- [14] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward. A new framework for distributed submodular maximization. *CoRR*, abs/1507.03719, 2015.
- [15] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *ICML*, pages 1236–1244, 2015.
- [16] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
- [17] P. Indyk, S. Mahabadi, M. Mahdian, and V. Mirrokni. Composable core-sets for diversity and coverage maximization. In *ACM PODS*, 2014.
- [18] Infochimps, Inc. English word list. <https://github.com/dwyl/english-words/blob/master/words.txt>.
- [19] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.
- [20] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in MapReduce and streaming. In *SPAA*, pages 1–10, 2013.
- [21] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *SPAA*, pages 85–94, 2011.
- [22] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Governance in social media: A case study of the Wikipedia promotion process. In *AAAI International Conference on Weblogs and Social Media*, 2010.
- [23] D. D. Lewis, Y. Yang, T. Rose, , and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.
- [24] V. S. Mirrokni and M. Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *STOC*, pages 153–162, 2015.

- [25] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. Lazier than lazy greedy. *arXiv preprint arXiv:1409.7938*, 2014.
- [26] B. Mirzasoleiman, A. Karbasi, A. Badanidiyuru, and A. Krause. Distributed submodular cover: Succinctly summarizing massive data. In *NIPS*, 2015.
- [27] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *NIPS*, pages 2049–2057, 2013.
- [28] B. Saha and L. Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SDM*, volume 9, pages 697–708. SIAM, 2009.
- [29] C. Wolff. Python package Gutenberg. <https://pypi.python.org/pypi/Gutenberg>. Version 0.4.2.
- [30] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, 2012.